Chapter

9

Nontraditional Optimization

Traditional methods have a tendency to be stuck in the local optimum point; mostly, they find the local or relative optimum point. Nontraditional methods are able to find the global optimization. These optimization algorithms are very useful to handle complicated problems in chemical engineering. In this chapter, we will discuss four methods namely Genetic Algorithm (GA), Particle Swarm Optimization (PWO), Simulated Annealing (SA), and Differential Evolution (DE) that are able to find the global optimizer. These stochastic algorithms are susceptible to premature termination. "Premature optimization is the root of all evil." – Donald Ervin Knuth (Art of Computer Programming, Volume 1: Fundamental Algorithms). Therefore, we should be very careful about the termination criteria of these stochastic optimization algorithms; otherwise, we will get wrong information from the optimization study.

9.1 Genetic Algorithm

Many useful optimum design problems in chemical engineering are described by mixed discrete–continuous variables, discontinuous and non-convex design spaces. When the traditional nonlinear programming methods are applied for these problems, they will be ineffective and computationally expensive. Mostly, they find a local (relative) optimum point that is nearby to the starting point. Genetic algorithms (GAs) are suitable for solving of this kind of problems, and in most instances, they are able to locate the global optimum solution with high accuracy. GAs are stochastic techniques whose search procedures are modeled similar to the natural evolution. Philosophically, Genetic algorithms work based on the theory of Darwin "Survival of the fittest" in which the fittest species will persist and reproduce while the less fortunate tend to disappear. To preserve the critical information, GAs encode the optimization problem to a chromosome-like simple data structure and employ recombination operators to these structures.

The GA was first proposed by Holland in 1975 [Holland, (1975)]. This approach works based on the similarity of improving a population of solutions by transforming their gene pool.

Two types of genetic modification, crossover, and mutation are utilized and the elements of the optimization vector, X, are expressed as binary strings. Crossover operation (Figs 9.1–9.2) deals with random swapping of vector elements (among parents that have highest objective function value or other ranking populations) or any linear combination of two parents. Mutation operation (Fig. 9.3) involved with the incorporation of a random variable to an element of the vector. The GAs have been used efficiently in process engineering, and numerous codes are readily available. For example, Edgar, Himmelblau, and Lasdon (2002) have been described a related GA algorithm that is available in Excel. Some case studies in process engineering consist of scheduling of batch process [Jung *et al.* (1998), Löeh *et al.* (1998)], and synthesis of mass transfer network [Garrard and Fraga, (1998)].

Unlike conventional optimization processes, GA is a robust, global and is not bound to domain-specific heuristics. GA is very useful for solving complex multi-objective optimization problems because

i. the objective function need not to be continuous and/or differentiable
ii. extensive problem formulation is not required
iii. they are less sensitive to the starting point (X^0)
iv. they generally do not get trapped into suboptimal local optima and
v. they are capable of finding out a function's true global optimum

These advantages of GAs increase their application to various fields [Babu and Angira (2006)].

9.1.1 Working principle of GAs

GAs are well suited for both maximization and minimization problems. The fitness function essentially measures the quality of each candidate solution and its magnitude is proportional the fitness of objective function.

GA for a constrained optimization problem can be represented as

$$\text{Maximize } f(X) \tag{9.1}$$

$$\text{subject to } h_j(X) = 0, \qquad j = 1, 2, \ldots q \tag{9.2a}$$

$$g_j(X) \le 0, \qquad j = q+1, \ldots m \tag{9.2b}$$

$$x_i^l \le x_i \le x_i^u, \qquad i = 1, 2, \ldots n \tag{9.2c}$$

GAs work based on the theories of natural genetics and natural selection. The essential elements of a natural genetics are employed in the genetic search process are reproduction, crossover, and mutation. The GA algorithm starts with a population of random strings representing the design variable. A fitness function is found to evaluate each string. The three main GA operators – reproduction, crossover, and mutation are employed to the random population to generate a new population. The new population is evaluated and tested until the termination criterion is met, iteratively altered by the GA operators. The term "generation" in GA represents the cycle of operation by the genetic operators and the evaluation of the fitness function. The steps of Genetic Algorithms are discussed below.

9.1.1.1 Initialization

Genetic Algorithms start with the initialization of a population of suitable size of chromosome length. All the strings are evaluated for their fitness values using specified fitness function. The objective function is interpreted as the problem of minimization and maximization and becomes the fitness function. They initiate with parent chromosomes selected randomly within the search space to generate a population. The population "evolves" towards the superior chromosomes by employing operators that are imitating the genetic processes taking place in the nature: selection or reproduction, recombination or crossover, and mutation.

Coding of GA

The representation of chromosomes (design variables) or coding of GA can be classified into two main categories: real and binary coding. In real coding, each variable is represented by a floating point number whereas binary coding transforms the variables within the chromosomes into a binary representation of zero and one before carrying out crossover and mutation operations. Both methods have some advantages and had been proven effective for certain problems [Herrera *et al.* (1998)]. The benefit of the binary representation is that conventional crossover and mutation operations become very simple. The drawback is that the binary numbers need to be transformed to real numbers when the design variable is to be estimated.

Example 9.1

Convert these decimal numbers into binary code

13, 19, 27, 30, and 46

Solution

The converted values are as follows

Table 9.1 Decimal to binary conversion

Sl. No	Decimal value	Binary code
1	13	001101
2	19	010011
3	27	011011
4	30	011110
5	46	101110

9.1.1.2 Reproduction

Selection process compares the chromosomes within the population intending to pick out those that will take part in the reproduction process. Reproduction selects good strings in a population and forms a mating pool. The reproduction operator will pick the above-average strings from the current population. The selection takes place with a specified probability on the basis of fitness functions. This fitness function plays a significant role to differentiate between bad and good solutions. The string that has a higher fitness value will represent a larger range in the cumulative probability values and consequently has a higher probability of being copied into the mating pool. Therefore, individuals that have higher fitness should have more chances to reproduce. Conversely, a string with a smaller fitness value signifies a smaller range in the cumulative probability values and has less probability of being copied into the mating pool.

The selection of members for the next generation could be done by the Roulette Wheel selection technique [Goldberg (1989)] in which the members with high fitness value have a higher probability of being replicated in the next generation.

Fitness evaluation

After defining the genetic representation, the next step is to find out the fitness functions related to the solutions. They are computed on the basis of the chromosomes phenotype. A fitness function evaluates the difference between bad and good solutions by mapping the solution within a non-negative interval. Two mappings corresponding to the maximization and minimization are as follows [Handbook of Evolutionary Computation (1997)]

i. During the minimization of problems, for some solution i, the fitness function is estimated according to the equation:

$$\text{fitness}\left(\vec{x}_i(t)\right) = \frac{1}{1 + f\left(\vec{x}_i(t)\right) - f_{\min}(t)}, \quad \vec{x}_i(t) \in X_x \tag{9.3}$$

where X_x represents a design space of the vector of control variables $\vec{x}_i$; and $f_{\min}$ is the minimum value obtained for the objective function up to generation t.

ii. For maximization problems, it estimates the fitness function by the following relation:

$$\text{fitness}\left(\vec{x}_i(t)\right) = \frac{1}{1 + f_{\max}(t) - f\left(\vec{x}_i(t)\right)}, \quad \vec{x}_i(t) \in X_x \tag{9.4}$$

where $f_{\max}$ denotes the maximum value attained for the objective function up to generation t.

Roulette-wheel selection

Roulette-wheel is the easiest method for proportionate selection. In this method, the individuals of each population are considered as slots of the Roulette-wheel. The width of each slot is proportional to the probability for selecting the corresponded chromosome. The scaled fitness function is applied to determine corresponding selection probabilities:

$$\Pr(i) = \frac{\text{fitness}(\vec{x}_i)}{\sum_{i=1}^{\mu} \text{fitness}(\vec{x}_i)} \tag{9.5}$$

where μ represents the population size.

In Roulette-wheel selection, emphasis is given to the better individuals within the population; therefore, a large pressure has been exerted on the search procedure. The probable number of copies in the sampling pool that some solution could obtain varies proportionally to its selection probability. This might cause loosing of genetic diversity, which leads to premature convergence to a suboptimal solution. Picking the samplings for reproduction is done by utilizing the so-called Roulette-wheel sampling algorithm. The process is repeated until the sampling pool is completed.

Elitism

Elitism is used to copy best parents to the next generation to replace worst offspring. In elitist selection, the best member in the population can be forwarded to the next generation without any modifications. When genetic operators are applied on the population, there is a chance of losing the best chromosome. Therefore, the best-fit chromosome is preserved at each generation. The fitness of offspring is compared with their parent chromosomes.

9.1.1.3 Crossover

The recombination (crossover) is carried out after completion of the selection process. In the crossover operation, new strings are created by exchanging information between strings of the mating pool. Crossover is the operation that ensure the genetic diversity of the population. Two strings involved in the crossover operation are recognized as parent strings whereas the resulting strings are identified as child strings. The child strings produced may be good or not, which depends on the performance of the crossover site. The outcome of crossover may be beneficial or detrimental. In order to conserve some good strings that are already existing in the mating pool, all strings in the mating pool are not used in the crossover.

The crossover operation can take place at a single point or at more than one point (multipoint) [Davis (1991)].

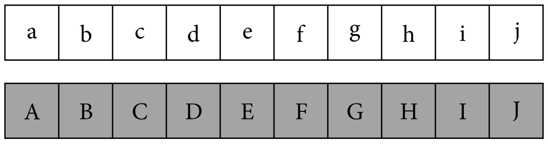

Fig. 9.1 Parent chromosomes

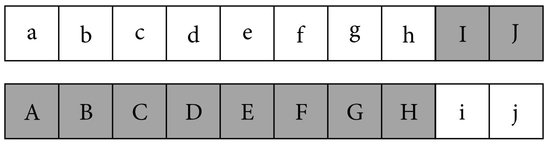

Fig. 9.2 Single point crossover

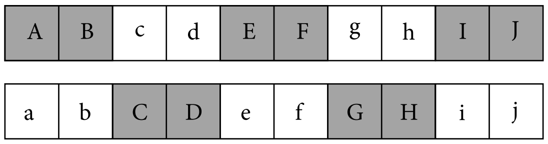

Fig. 9.3 Multipoint crossover

Example 9.2

Perform crossover operation between the 1–2 and 3–4 binary codes of Example 9.1.

Solution

The binary codes of Example 9.1 are

Table 9.2 Single and multiple point crossover

Sl. No	Binary code
Single point crossover	
1	001101
2	010011
Multiple point crossover	
3	011011
4	011110

after crossover, the chromosomes becomes as in Table 9.3

Table 9.3 Chromosomes after crossover

Sl. No	Binary code	Decimal value
1	010101	21
2	001011	11
3	011010	26
4	011111	31

9.1.2.4 *Mutation*

After recombination operation, offspring undergoes to mutation process (Fig. 9.4). The mutation operation is the creation of a new chromosome from one and only one individual with a predefined probability. Mutation is required to generate a point in the neighborhood of the current point, thus, achieving local search around the current solution. This mutation operation is also used to preserve diversity in the population.

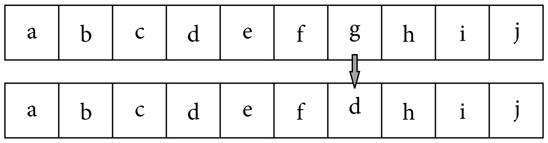

Fig. 9.4 Mutation operation

After completion of these three operations, the offspring is inserted into the population that produces a new generation by replacing the parent chromosomes from which they were derived (see Fig. 9.5). This cycle is carried out until the optimization criterion is met [Shopova and Vaklieva-Bancheva (2006)]. Theoretically, there is no clear explanation for the distinction between generations. This restriction is merely an implementation model that makes the computation straightforward. During the search, to facilitate preservation of a good solution, sometimes we employ an elitism procedure, i.e., the best individuals from the old population are directly copied into the new one. This procedure ensures that the overall solution of the GA will not become worse.

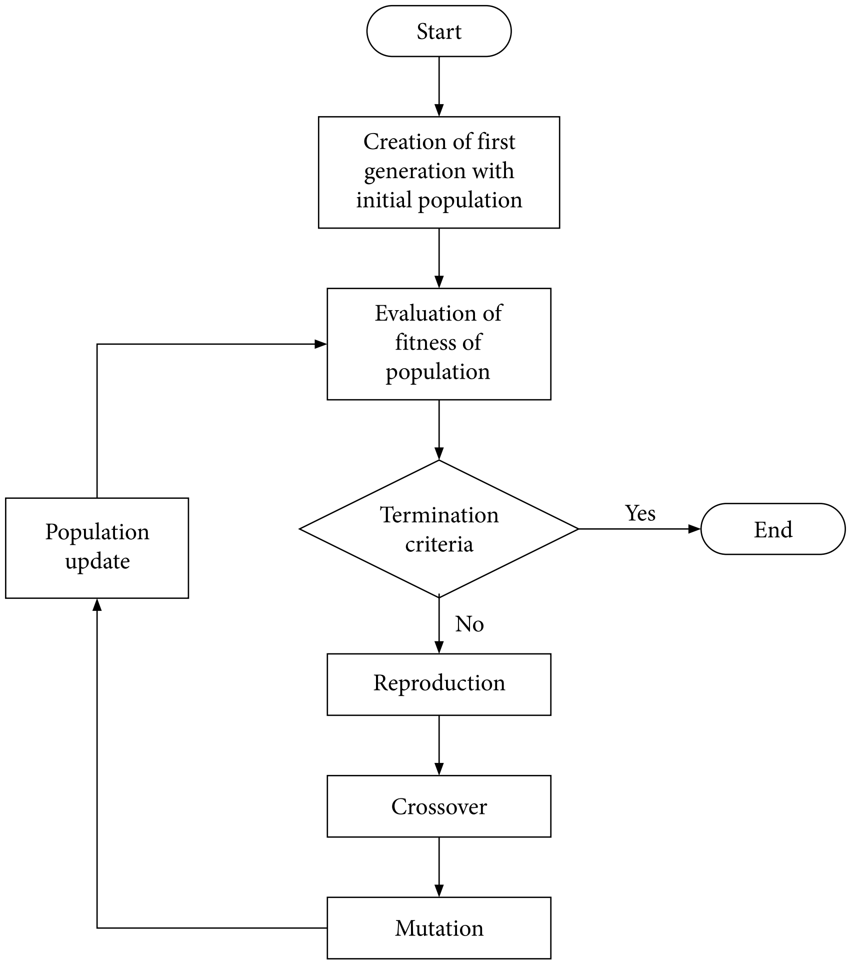
Start
Creation of first generation with initial population
Evaluation of fitness of population
Termination criteria
Yes
End
No
Reproduction
Crossover
Mutation
Population update

Fig. 9.5 Flowchart of genetic algorithm

9.1.2 Termination

Unlike simple neighborhood search techniques that terminate when they reach a local optimum, GAs are stochastic search methods that could in principle run forever. However, a termination criterion is required for practical application. The standard practices are setting a limit on the number of fitness evaluations or the clock time of computer, or tracking the diversity of the population and stop when this diversity falls below a predetermined threshold.

The significance of diversity in the latter case is not always clear, and it could relate to either the genotype or the phenotype or even, possibly, to the fitnesses. However, the most common way, of measuring this, is by genotype statistics. For instance, we could make a decision to terminate a run when at every locus the proportion of one particular allele raised above 90 per cent.

9.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based stochastic search method which is proposed by Kennedy and Eberhart in 1995 [Kennedy and Eberhart (1995)]. This is an evolutionary technique similar to genetic algorithm. Population based searches may be a useful alternative when the search space is very large and we need exhaustive search. However, population based search techniques do not guarantee you the best (optimal) solution. The PSO method was originally devised based on the social behaviour of a bird flocks, school of fish etc. In PSO, each member of the population is called a particle and the population is called a swarm. A swarm of particles moves around in the search-space influenced by the improvements identified by the other particles. This method is very helpful when calculation of the gradient is too laborious or even impossible to derive. This technique can be readily employed for a wide range of optimization problems because it does not utilize the gradient of the objective function to be optimized. The original PSO algorithm has some drawbacks. This algorithm does not work well continuously and it may require tuning of its behavioural parameters such that it will perform well on the problem at hand. Numerous research works have been carried out to improve the performance of the PSO method. A conventional and easy way of improving the PSO algorithm is by manually adjusting its behavioural parameters. Some researchers [Clerc and Kennedy (2002); I.C. Trelea (2003)] have done mathematical analysis to show how the contraction of the particles to a single point is influenced by its behavioural parameters.

Both Particle Swarm Optimizers and Genetic Algorithms have some common feature; both initialize a population in a similar fashion, both utilize a fitness function to determine potential solution and both are generational, that is both repeat the same set of processes for a predetermined amount of time. However, PSO is somewhat better than genetic algorithm (GA) for solving constrained optimization problems, because GA, which has been generally employed for solving such problems has disadvantage of slow convergence due to mutation operator leading to demolition of good genes consequently poor convergence. Particle Swarm Optimization is a computational intelligence-based method that is not considerably affected by the size and nonlinearity of the optimization problem. When most analytical methods fail to converge, PSO can converge to the optimal solution for some problems. As compared with other optimization methods, it is faster, cheaper and more efficient. It requires less memory space and lesser speed of CPU. In addition, adjustment of few parameters is required in PSO. For this reason, PSO is a useful optimization method for practical application. This method is well suited to solve the problems such as non-linear, non-convex, continuous, discrete, integer variable etc.

9.2.1 Working principle

In PSO method, particles move through the search space using an equation (Eq. (9.6)) which is a combination of the individual best solution and the best solution that has found by any particle in their neighborhood. The neighborhood of an individual particle is defined as the subset of particles with which it can communicate. For a particle communication, the first PSO model used a Euclidian neighborhood where the actual distance between particles are measured to determine which were closer to be in communication. This was performed by imitating the behavior of birds flocks, like the biological models where individual birds are only able to communicate with other birds in the immediate vicinity [Reynolds (1987); Heppner and Grenander (1990)].

PSO is an evolutionary computation technique, which was modeled similar to the social behavior of fish schooling, and bird flocking. PSO algorithm is motivated from the artificial life and social psychology, as well as engineering and computer science. It uses a "population" of particles that travel through the hyperspace of the problem with specified velocities. The velocities of the individual particles are adjusted stochastically at each iteration. The historical best position of the particle itself and the neighborhood best position are used for this purpose. Both the particle best and the neighborhood best are determined by considering a user defined fitness function. The movement of each particle usually results in an optimal or near-optimal solution. The word "swarm" originates from the random movements of the particles in the problem space, now it is more comparable to a swarm of mosquitoes rather than a school of fish or a flock of birds. The PSO can be explained by Fig. 9.6. There are three velocity components cognitive velocity V^t_{pbest}, social velocity V^t_{gbest} and inertia velocity V^t_i. Particle moves from initial position X^t_i to the new position X^{t+1}_i. The new velocity can be calculated using Eq. (9.7).

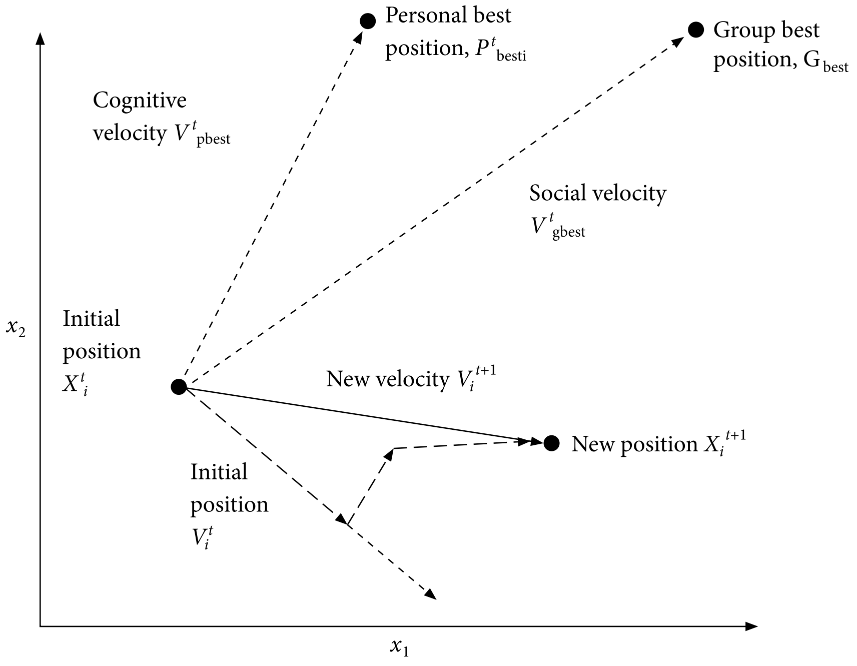
Personal best position, P^t_{besti}
Group best position, G_{best}
Cognitive velocity V^t_{pbest}
Social velocity V^t_{gbest}
x_2
Initial position X^t_i
New velocity V_i^{t+1}
New position X_i^{t+1}
Initial position V^t_i
x_1

Fig. 9.6 Movement of particles in PSO

The PSO algorithm includes some tuning parameters that significantly influence the performance of the algorithm, often stated as the exploration–exploitation tradeoff. The capability of a search algorithm to examine the diverse region of the search space in order to establish a good optimum is called exploration. In contrast, exploitation is the capability to focus the search around a promising region to facilitate refinement of a candidate solution [Ghalia (2008)]. Using their exploration and exploitation, the particles of the swarm fly through hyperspace and have two essential reasoning capabilities: their memory of their own best position – personal best (P_{best}) and knowledge of the global or their neighborhood's best – global best (G_{best}). Position of the particle is determined by its velocity. Let X_i^t represent the position of particle i within the search space at time step t; unless otherwise stated, t indicates discrete time steps. The updated position of the particle is determined by adding a velocity V_i^t to the current position.

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{9.6}$$

where

$$V_i^t = V_i^{t+1} + c_1 r_1 \left(P_{\text{best}}^t - X_i^{t-1}\right) + c_2 r_2 \left(G_{\text{best}}^t - X_i^{t-1}\right) \tag{9.7}$$

The PSO algorithm is similar to GA as this system is also initialized with a population of random solutions. The main difference with GA is that; in PSO, a randomized velocity is allotted to each potential solution, and the probable solutions, called particles, are then traveled through the problem space.

9.2.2 Algorithm

The steps for implementing the PSO are as follows:

Step 1 Initialize a population of particles (array) with random position and velocities on D-dimensional problem space.

Step 2 For each particle, estimate the desired optimization fitness function in D variables.

Step 3 Compare the fitness of particles at current position with particle's P_{best}. If current value is better than P_{best}, then replace current value with P_{best} value, then the P_{best} location turns into the current location in D-dimensional space.

Step 4 Compare values of fitness function with the overall previous best of the population. If current value is better than G_{best}, then reset G_{best} to the current particle's array index and value.

Step 5 Modify the position and velocity of the particle as expressed by Eqs (9.6) and (9.7) respectively.

Step 6 Go to Step 2 until a criterion is met. Usually a satisfactorily good fitness or a maximum number of iterations (generations) are considered as termination criteria.

9.2.3 Initialization

In PSO algorithm, initialization of the swarm is very important because proper initialization may control the exploration and exploitation tradeoff in the search space more efficiently and find the better result. Initially the particles are placed at arbitrary locations within the search-space and then moving in randomly defined directions. Usually, a uniform distribution over the search space is used for initialization of the swarm. The initial diversity of the swarm is important for PSO's performance, it denotes that how well particles are distributed and how much of the search space is covered. Therefore, when the initial swarm does not cover the entire search space, the PSO algorithm will have difficultly to find the optimum if the optimum is located outside the covered area. Then, the PSO will only discover the optimum if a particle's momentum takes the particle into the uncovered area. Therefore, the optimal initial distribution to be located within the domain defined by $X_{\min}$ and $X_{\max}$ which represent the minimum and maximum ranges of X for all particles i in dimension j respectively. Then the initialization method for the position of each particle is given by

$$X^0 = X_{\min,j} + r_j\left(X_{\max,j} - X_{\min,j}\right) \tag{9.8}$$

where $r_j \sim U(0,1)$

The selected size of the population depends on the problem considered. The most common population size is 20–50. It is found that smaller populations are common for evolutionary algorithms like evolutionary programming and genetic algorithms. It was optional for PSO in terms of minimizing the total number of evaluations (size of population times the number of generations) required to achieve a satisfactory solution.

The velocities of the particles can be initialized to zero, i.e., $V_i^0 = 0$, since randomly initialized particle's positions already ensure random positions and moving directions. Then, the direction of a particle is changed gradually hence, it will start moving in the direction of the best previous positions of itself and its peers, searching in their surrounding area and hopefully finding even better positions with reference to some fitness measure $f : \mathbb{R}^n \to \mathbb{R}$. The trajectory of every particle is determined by a simple rule combining the current velocity and exploration histories of the particle and its neighbors. The particles may also be initialized with nonzero velocities, however, it must be done with care and such velocities must not be very large. In general, large velocity has large momentum and accordingly large position update. Therefore, such large initial position updates can cause particles to move away from boundaries of the feasible region, and the algorithm needs to consider more iteration before settling the best solution.

9.2.4 Variants of PSO

There are many variants of PSO, which are developed to overcome the limitations of the original PSO algorithm. Originally, the inertia weight coefficient was not an element of the PSO algorithm, however, later it was modified that became commonly accepted. Some PSO variants do not comprise a single global best characteristic of the algorithm; rather they utilize multiple global best that are provided by separate subpopulations of the particles. In addition, PSO is also influenced by velocity clamping, and velocity constriction and these parameters are described in the following sections.

9.2.4.1 PSO with inertia weight

In 1998, Shi and Eberhart came up with a variant of PSO called PSO with inertia. A nonzero inertia weight introduces a choice for the particle for continuing their movement in the same direction it was moving on the earlier iteration. The utilization of inertia weight (w) has shown better performance in several applications. In the beginning when the method was developed, the value of w often reduced linearly from about 0.9 to 0.4 during a run. Proper selection of the inertia weight gives a balance between global and local exploration and exploitation, and results in less number of iteration on average to find a satisfactorily optimal solution. Generally, lower values of the inertial coefficient expedite the convergence of the swarm to the optimum point whereas higher values of the inertial coefficient promote exploration of the search space entirely. Declining inertia with time begins a shift from the exploratory (global search) to the exploitative (local search) mode.

To control the balance between the global and local exploration, to achieve quick convergence, and to attain an optimum, value of the inertia weight is decreasing according to the following equation

$$\omega^{t+1} = \omega_{\max} - \left(\frac{\omega_{\max} - \omega_{\min}}{t_{\max}}\right)t, \quad \omega_{\max} > \omega_{\min} \tag{9.9}$$

$\omega_{\max}$ and $\omega_{\min}$ are the initial and final values of the inertia weight respectively

$t_{\max}$ is the maximum iteration number and t is the current iteration number

Equations (9.11) and (9.12) express the velocity and position update equations integrated with an inertia weight. It can be observed that these equations are identical to Eqs (9.6) and (9.7) with the addition of the inertia weight ω as a multiplying factor of V_i^{t+1}.

$$X_i^t = f\left(\omega^t, X_i^{t-1}, \mathrm{V}_i^{t-1}, P_{\text{best}}, G_{\text{best}}\right) \tag{9.10}$$

where

$$V_i^t = \omega^t V_i^{t+1} + c_1 r_1 \left(P_{\text{best}}^t - X_i^{t-1}\right) + c_2 r_2 \left(G_{\text{best}}^t - X_i^{t-1}\right) \tag{9.11}$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{9.12}$$

9.2.4.2 PSO with constriction coefficient

Clerc [Clerc, (1999)] developed the PSO with Constriction Coefficient in 1999. Particle swarms interactions are represented by a set of complex linear equations. Utilization of constriction coefficient results in particle convergence over time. That is the amplitude of the particle's oscillations reduces as it focuses on the local and neighborhood previous best positions. Although the particle will converge to a point in due course, the constriction coefficient also prevents collapse if the right social circumstances are prepared. The particle oscillates around the weighted mean of P_{best} and G_{best}, when the previous best position and the neighborhood best position are close to each other the particle will conduct a local search. The particle will carry out a global search (more exploratory search) when the previous best position and the neighborhood best position are distant from each other. During this search process, the neighborhood best position and previous best position will be modified and the particle will be shifted from local search back to global search. Therefore, the constriction coefficient method makes a balance of the necessity for global and local search depending on whatever social circumstances are in place.

$$X_i^t = f\left(K, X_i^{t-1}, \mathrm{V}_i^{t-1}, P_{\text{best}}, G_{\text{best}}\right) \tag{9.13}$$

$$K = \frac{2k}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad \text{where } \varphi = c_1 + c_2, \ \ \varphi > 4 \tag{9.14}$$

$$V_i^t = K\left[V_i^{t-1} + c_1\varphi_1\left(P_{\text{best}} - X_i^{t-1}\right) + c_2\varphi_2\left(G_{\text{best}} - X_i^{t-1}\right)\right] \tag{9.15}$$

$$X_i^t = X_i^{t-1} + V_i^t \tag{9.16}$$

By modifying φ, we can control the convergence characteristics of the system. Usually, $k = 1$ and $c_1 = c_2 = 2$, and φ is set to 4.1, thus, $K = 0.73$.

In PSO algorithm, all of these three parts of the velocity update equation (Eq. (9.15)) have different functions. The first part $\omega^t \times V_i^{t+1}$ is the inertia component, which is accountable for keeping the particle moving in the same direction it was originally traveling. The second part $c_1 r_1\left(P_{\text{best}}^t - X_i^{t-1}\right)$, called the cognitive component, acts as memory of the particle, causes to be inclined to return to the regions of the search space within which it has experienced higher individual fitness. The third part $c_2 r_2\left(G_{\text{best}}^t - X_i^{t-1}\right)$, which helps the particle to move toward the best region the swarm has encountered so far is called the social component.

9.2.4.3 Velocity clamping

The problem with basic PSO is that the particles velocities quickly explode to large values. Under some setting, the swarm can become scattered in a very wide area in the search domain, resulting in a very high particle velocity. In order to avoid such an explosion, a practice has been followed to limit the maximum particle velocity component as V_{max}. Eberhart and Kennedy first proposed the velocity clamping; it helps particles to stay inside the boundary and to take reasonable step size so as to examine through the search space. Without this velocity clamping in the searching space, the process will be likely to explode and positions of the particles change rapidly [Bratton and Kennedy, (2007)]. Whenever a search space is bounded by the range $[-X_{\text{max}}, X_{\text{max}}]$, velocity clamping confines the velocity to the range $[-V_{\text{max}}, V_{\text{max}}]$, where $V_{\text{max}} = k \times X_{\text{max}}$. The value k signifies a user-defined velocity clamping factor, $1.0 \geq k \geq 0.1$. During some optimization processes, the search space is not centered around 0 and therefore this $[-X_{\text{max}}, X_{\text{max}}]$ range is not an satisfactory definition of the search space. In such a case where the search space is bounded by the range, $[X_{\text{min}}, X_{\text{min}}]$ we define $V_{\text{max}} = k \times (X_{\text{max}} - X_{\text{min}})/2$.

If the maximum velocity $V_{\max}$ is too large, then the particles may move erratically and jump over the optimal solution. On the other hand, if $V_{\max}$ is too small, the particle's movement is limited and the swarm may not explore sufficiently or the swarm may become trapped in a local optimum.

The velocity update rule with this saturation is given by:

$$V_{i,j}^{t+1} = \begin{cases} V_{i,j}^{t+1} & \text{if } \left|V_{i,j}^{t+1}\right| < V_{\max,j} \\ V_{\max} & \text{if } \left|V_{i,j}^{t+1}\right| \geq V_{\max,j} \end{cases} \tag{9.17}$$

Once, the velocity for each particle is determined, the position each particle is updated by applying the new velocity to the particle's previous position. This process is repeated until it satisfied some stopping criteria.

9.2.5 Stopping criteria

Stopping criteria is used to terminate the iterative search process. Some stopping criteria are discussed below:

1. The algorithm is terminated when a maximum number of iterations or function evaluations (FEs) have been reached. If this maximum number of iterations (or FEs) is very small, the search process may stop before a good result has been obtained.
2. The algorithm is terminated when there is no significant improvement over a number of iterations. This improvement can be measured in different ways. For instance, the process may be considered to have terminated whenever the average change of the particles' positions are too small or the average velocity of the particles is approximately zero over a number of iterations.
3. The algorithm is terminated when the normalized swarm radius is approximately zero. The normal swarm radius is defined as

$$R_{\text{norm}} = \frac{R_{\max}}{\text{diameter}(S)} \tag{9.18}$$

where diameter(S) is the initial swarm's diameter and $R_{\max}$ is the maximum radius,

$$R_{\max} = \left\| x_m - G_{\text{best}} \right\| \tag{9.19}$$

with

$$\left\| x_m - G_{\text{best}} \right\| \geq \left\| x_i - G_{\text{best}} \right\|, \quad m, \forall i = 1, 2, \ldots n \tag{9.20}$$

and $\|\bullet\|$ is a suitable distance norm.

The process will terminate when $R_{\text{norm}} < \varepsilon$. If ε is very large, the process can be terminated prematurely before a good solution has been reached while if ε is too small, the process may need more iterations.

9.2.6 Swarm communication topology

PSO algorithm relies on the social interaction among the particles in the entire swarm. A neighborhood of each particle is required to define properly because this neighborhood determines the extent of social interaction within the swarm and influences a particular particle's movement. Particles communicate with one another by exchanging information about the success of each particle in the swarm. When a particle in the whole swarm finds a better position, all particles move towards this particle. This performance of the particles is determined by the particles' neighborhood. When the neighborhoods in the swarm are small, less interaction occurs between individuals. For small neighborhood, the convergence is slower however, it improves the quality of solutions. Whereas for larger neighborhood, the convergence will be faster however, sometimes convergence may occurs earlier. Therefore, in practice, the search process starts with small neighborhoods size and then increases with time. The performance of PSO may be improved by designing different types of neighborhood structures. Some neighborhood structures or topologies are discussed below:

Five major neighborhood topologies have been used in PSO: Von Neumann, star, wheel, circle and pyramid [Valle *et al.* (2008)]. The selection for neighborhood topology decides which individual to use for G_{best}. In the circle topology, each individual in socially connected to its k nearest topological neighbors.

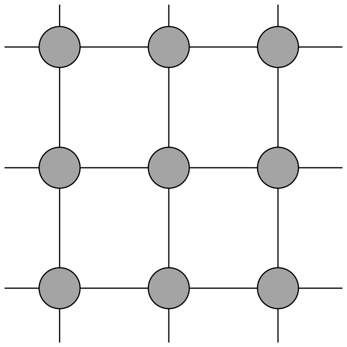

Fig. 9.7a von Neumann topology

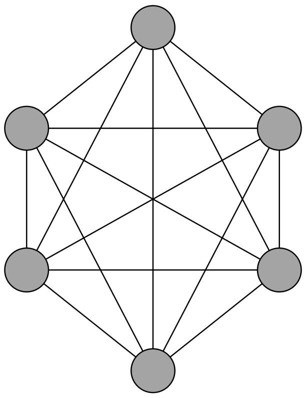

Fig. 9.7b Star topology

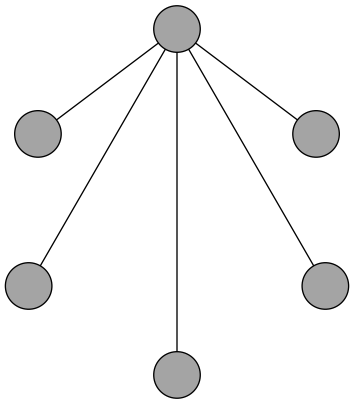

Fig. 9.7c Wheel topology

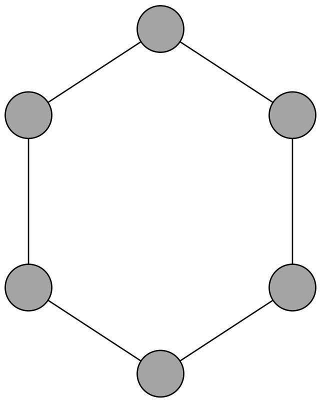

Fig. 9.7d Circle topology

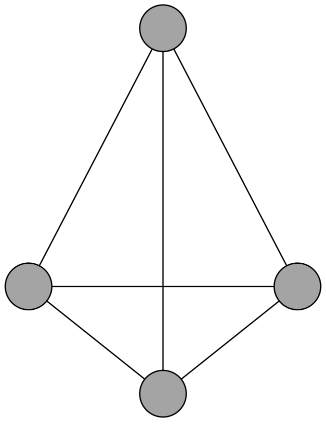

Fig. 9.7e Pyramid topology

Kennedy suggested that the G_{best} version [Fig. 9.7b] converges fast however, may be trapped in a local minimum, while the P_{best} network has more chances to find an optimal solution, although it converges slowly.

Example 9.3

Solve problem 3.3 using PSO, show only two iterations.

Solution

The objective function can be written as follows

$$W = 301.8\left[(x)^{0.286} + \left(\frac{10}{x}\right)^{0.286} - 2\right]$$

consider $0 \le x \le 12$

the initial population for iteration number $t = 0$

$$x_1^0 = 1.0,\ \ x_2^0 = 3.0,\ \ x_3^0 = 5.0,\ \ x_4^0 = 7.0,\ \ x_5^0 = 9.0,\ \ x_6^0 = 12.0$$

$$f_1^0 = 281.268038,\ f_2^0 = 235.470634,\ f_3^0 = 242.587797,\ f_4^0 = 257.134263,$$

$$f_5^0 = 273.193037,\ f_6^0 = 297.144363$$

let $c_1 = c_2 = 2$

set the initial velocities of each particle to zero:

$v_i^0 = 0$, i.e., $v_1^0 = v_2^0 = v_3^0 = v_4^0 = v_5^0 = v_6^0 = 0$

Step 2 Set iteration number $t = 0 + 1 = 1$

Step 3 Find the personal best for each particle by

$$P_{\text{best},i}^{t+1} = \begin{cases} P_{\text{best},i}^{t} & \text{if } f_i^{t+1} > f_i^t \\ X_i^{t+1} & \text{if } f_i^{t+1} \le f_i^t \end{cases}$$

so,

$$P_{\text{best},1}^1 = 1,\ P_{\text{best},2}^1 = 3,\ P_{\text{best},3}^1 = 5,\ P_{\text{best},4}^1 = 7,\ P_{\text{best},5}^1 = 9,\ P_{\text{best},6}^1 = 12$$

Step 4 Find the global best by

$$G_{\text{best}} = \min\left\{P_{\text{best},i}^t\right\} \text{ where } i = 1,2,3,4,5,6$$

Since, the minimum personal best is $P^1_{\text{best},2} = 3$, thus, $G_{\text{best}} = 3$

Step 5 Consider the random numbers in the range (0,1) as $r_1^1 = 0.512$, $r_2^1 = 0.796$, $\omega_{\max} = 0.9$, $\omega_{\min} = 0.4$ and maximum iteration number 10.

so, $\omega^1 = \omega_{\max} = 0.9$

and find the velocities of the particles by

$$v_i^{t+1} = \omega^{t+1} v_i^t + c_1 r_1^t \left[P^t_{\text{best},i} - X_i^t\right] + c_2 r_2^t \left[G^t_{\text{best},i} - X_i^t\right];\ i = 1,2,\ldots,6$$

so,

$$v_1^1 = 0 \times 0.9 + 2 \times 0.512(1-1) + 2 \times 0.398(3-1) = 1.592$$

$$v_2^1 = 0 \times 0.9 + 2 \times 0.512(3-3) + 2 \times 0.398(3-3) = 0.0$$

$$v_3^1 = 0 \times 0.9 + 2 \times 0.512(5-5) + 2 \times 0.398(3-5) = -1.538$$

$$v_4^1 = 0 \times 0.9 + 2 \times 0.512(7-7) + 2 \times 0.398(3-7) = -3.184$$

$$v_5^1 = 0 \times 0.9 + 2 \times 0.512(9-9) + 2 \times 0.398(3-9) = -4.776$$

$$v_6^1 = 0 \times 0.9 + 2 \times 0.512(12-12) + 2 \times 0.398(3-12) = -7.164$$

Step 6 Find the new values of $x_i^1,\ i = 1,\ldots,6$

$$x_1^1 = x_1^0 + v_1^1 = 1 + 1.592 = 2.592$$

$$x_2^1 = x_2^0 + v_2^1 = 3 + 0.0 = 3.0$$

$$x_3^1 = x_3^0 + v_3^1 = 5 - 1.538 = 3.462$$

$$x_4^1 = x_4^0 + v_4^1 = 7 - 3.184 = 3.816$$

$$x_5^1 = x_5^0 + v_5^1 = 9 - 4.776 = 4.224$$

$$x_6^1 = x_6^0 + v_6^1 = 12 - 7.164 = 4.836$$

now calculate the values of the objection function (fitness function)

$$f_1^1 = 236.732707,\ f_2^1 = 235.470634,\ f_3^1 = 235.656786,\ f_4^1 = 236.587280,$$

$$f_5^1 = 238.252585,\ f_6^1 = 241.574742$$

Now we will move to next iteration $t = 1 + 1 = 2$

Step 7 Find the personal best for each particle by

$$P_{\text{best},i}^{t+1} = \begin{cases} P_{\text{best},i}^{t} & \text{if } f_i^{t+1} > f_i^t \\ X_i^{t+1} & \text{if } f_i^{t+1} \le f_i^t \end{cases}$$

so,

$$P_{\text{best},1}^2 = 1.592,\ P_{\text{best},2}^2 = 3,\ P_{\text{best},3}^2 = 3.462,\ P_{\text{best},4}^2 = 3.816,\ P_{\text{best},5}^2 = 4.224,\ P_{\text{best},6}^2 = 4.836$$

The minimum personal best is $P_{\text{best},2}^2 = 3$, thus, $G_{\text{best}} = 3$

Consider the random numbers in the range (0,1) as $r_1^2 = 0.371,\ r_2^1 = 0.602$

$$\omega^2 = 0.9 - \frac{(0.9-0.4)}{10} \times 1 = 0.85$$

$$v_1^2 = 1.592 \times 0.85 + 2 \times 0.371(2.592 - 2.592) + 2 \times 0.602(3 - 2.592) = 1.7612$$

$$v_2^2 = 0.0 \times 0.85 + 2 \times 0.371(3.0 - 3.0) + 2 \times 0.602(3.0 - 3.0) = 0.0$$

$$v_3^2 = (-1.538) \times 0.85 + 2 \times 0.371(3.462 - 3.462) + 2 \times 0.602(3 - 3.462) = -1.86355$$

$$v_4^2 = (-3.184) \times 0.85 + 2 \times 0.371(3.816 - 3.816) + 2 \times 0.602(3 - 3.816) = -3.68886$$

$$v_5^2 = (-4.776) \times 0.85 + 2 \times 0.371(4.224 - 4.224) + 2 \times 0.602(3 - 4.224) = -5.533296$$

$$v_6^2 = (-7.164) \times 0.85 + 2 \times 0.371(4.836 - 4.836) + 2 \times 0.602(3 - 4.836) = -8.299944$$

Find the new values of $x_i^2,\ i = 1,\ldots,6$

$$x_1^2 = x_1^1 + v_1^2 = 2.592 + 1.7612 = 4.3532$$

$$x_2^2 = x_2^1 + v_2^2 = 3 + 0.0 = 3.0$$

$$x_3^2 = x_3^1 + v_3^2 = 3.462 - 1.86355 = 1.59845$$

$$x_4^2 = x_4^1 + v_4^2 = 3.816 - 3.68886 = 0.12714$$

$$x_5^2 = x_5^1 + v_5^2 = 4.224 - 5.533296 = -1.309296 \quad \text{(Beyond the range)}$$

$x_6^1 = x_6^0 + v_6^1 = 4.836 - 7.164 = -2.328$ (Beyond the range)

Now we have to calculate f_i^2 at these new x_i^2 points. After sufficient number of iteration, the result will converge.

9.3 Differential Evolution

Differential evolution (DE) is a stochastic, population-based optimization technique which was introduced by Storn and Price in 1996 [Storn and Price (1997)]. DE is a powerful yet simple evolutionary algorithm (EA) for optimizing real-valued, multimodal functions [Price (1996)]. The EAs are encouraged by the natural evolution of species, have been effectively implemented in the field of optimization. However, user needs to choose the suitable parameters during the implementation of EAs. Improper selection of parameter setting may lead to high computational costs due to the time-consuming trial-and-error parameter and operator tuning process. The success of DE process to solve a particular problem significantly depends on correctly selection of trial vector generation strategies and their related control parameter values. Therefore, for solving a particular optimization problem at hand successfully, it is usually required to make a trial-and-error search for the most suitable strategy and to adjust the values of the related parameter. However, this trial-and-error searching process is time-consuming and needed high computational costs. Furthermore, as evolution continued, the population of DE may travel through different regions in the search space, within which some strategies related with specific parameter settings may be more effectual than others may. Therefore, it is required to adaptively determine a proper strategy and its related parameter values at different stages of evolution/search process [Qin *et al.* (2009)].

In DE, there are numerous trial vector generation strategies out of which a few may be suitable to solve a particular problem. The optimization performance of the DE will be significantly influenced by three crucial control parameters involved in DE namely population size (*NP*), scaling factor (*F*), and crossover rate (*CR*). The DE algorithm has an advantage over other EAs; since DE is inherently parallel, further significant acceleration can be accomplished if the algorithm is carried out on a parallel machine or a network of computers. This is especially correct for real world applications where calculating the objective function involves a considerable amount of time.

9.3.1 DE algorithm

DE algorithm attempts to evolve a population of *NP* *D*-dimensional parameter vectors commonly known as individuals, which encode the candidate solutions, i.e., $X_{i,G} = \left\{x_{i,G}^1, \ldots, x_{i,G}^D\right\}$, $i = 1, \ldots NP$ towards the global optimum. The initial population should cover the entire search space as much as possible by randomizing individuals uniformly within the search space constrained by the prescribed minimum and maximum parameter bounds $X_{\min} = \left\{x_{\min}^1, \ldots, x_{\min}^D\right\}$ and $X_{\max} = \left\{x_{\max}^1, \ldots, x_{\max}^D\right\}$. For example, the initial value of the *j*th parameter in the *i*th individual at the generation $G = 0$ is generated by

$$x_{i,0}^j = x_{\min}^j + \text{rand}(0,1)\left(x_{\max}^j - x_{\min}^j\right), \quad j = 1, 2, \ldots, D \tag{9.21}$$

where rand(0, 1) represents a uniformly distributed random variable within the range [0,1]

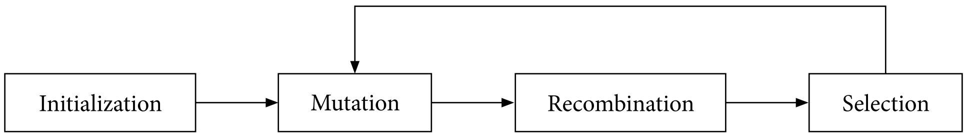

Fig. 9.8 Flowchart for differential algorithm

Figure 9.8 is the flowchart for DE method. This method starts with an initial population, which is generated randomly.

9.3.2 Initialization

Initialization is the first step of the DE algorithm. We need to define upper and lower bounds for each parameter:

$$x_{\min}^{j} \le x_{i,0}^{j} \le x_{\max}^{j} \tag{9.22}$$

After that, randomly select the initial parameter values uniformly on the intervals $\left[x_{\min}^{j}, x_{\max}^{j}\right]$ using the following equation.

$$x_{i,0}^{j} = x_{\min}^{j} + \text{rand}(0,1)\left(x_{\max}^{j} - x_{\min}^{j}\right), \; j = 1,2,\ldots,D \tag{9.23}$$

Population Size is a crucial parameter for DE algorithm.

Price and Storn (1997) suggested a minimum population size of 4 times the dimensionality (D) of the problem and 5–10 times is considered as the most suitable population size. In simulating crystal structures via DE, a population of 40 for a 7-dimensional problem (a factor of 5.7) is used by Weber and Bürgi [Weber and Bürgi (2002)]. These values have surely been shown to work nicely in practice, confirming that adequate genetic material is contained in the populations [Storn and Price, (1997)]. In Example 9.4 a population of 20 has been considered for a 3-dimensional problem (a factor of 6.67). However, this could be using an excessive amount research with other evolutionary algorithms (also using real-value coding) has provided most excellent results with factors between 1.5 and 2 [Mayer, (2002)]. Values within this range could be more effective, by carrying only a sufficient number of population members.

Example 9.4

Create the initial population for the function given below

$$f(X) = 1200(2x_1x_3 + 2x_2x_3) + 2500x_1x_2 + 500\left(\frac{1000}{x_1x_2x_3}\right) + 100\left(\frac{1000}{10x_1x_2x_3}\right)$$

The limit for variables are as follows : $0 \le x_1, x_x, x_3 \le 10$

Solution

Here, we will consider population size 20.

Therefore, for first generation $G = 0$, we have factor $j = 1, 2, 3$ and individual $i = 1, 2, \ldots, 20$.

Consider any random number, rand (0, 1)

From Eq. (9.23), for $i = 1, j = 1, 2, 3$

$$x_{1,0}^{1} = 0.0 + 0.638(10 - 0.0) = 6.38$$

$$x_{1,0}^{2} = 0.0 + 0.285(10 - 0.0) = 2.85$$

$$x_{1,0}^{3} = 0.0 + 0.819(10 - 0.0) = 8.19$$

Therefore, the first individual is [6.38,2.85,8.19].

Now for $i = 2, j = 1, 2, 3$

$$x_{2,0}^{1} = 0.0 + 0.122(10 - 0.0) = 1.22$$

$$x_{2,0}^{2} = 0.0 + 0.493(10 - 0.0) = 4.93$$

$$x_{2,0}^{3} = 0.0 + 0.720(10 - 0.0) = 7.20$$

Therefore, the second individual is [1.22,4.93,7.20].

Similarly, we have to find 20 individuals for generation $G = 0$.

9.3.3 Mutation

After initialization, DE employs the mutation operation in the current population to generate a mutant vector $V_{i,G}$ with respect to each individual $X_{i,G}$, so-called target vector. For each target vector $X_{i,G}$, at the generation G, its associated mutant vector $V_{i,G} = \left\{v_{i,G}^{1}, v_{i,G}^{2}, \ldots, v_{i,G}^{D}\right\}$ can be generated via certain mutation strategy. For example, the five most frequently used mutation strategies implemented in the DE codes are listed as follows:

$$V_{i,G} = X_{r_1^i,G} + F\left(X_{r_2^i,G} - X_{r_3^i,G}\right) \tag{9.24a}$$

$$V_{i,G} = X_{\text{best},G} + F\left(X_{r_1^i,G} - X_{r_2^i,G}\right) \tag{9.24b}$$

$$V_{i,G} = X_{i,G} + F\left(X_{\text{best},G} - X_{i,G}\right) + F\left(X_{r_1^i,G} - X_{r_2^i,G}\right) \tag{9.24c}$$

$$V_{i,G} = X_{\text{best},G} + F\left(X_{r_1^i,G} - X_{r_2^i,G}\right) + F\left(X_{r_3^i,G} - X_{r_4^i,G}\right) \tag{9.24d}$$

$$V_{i,G} = X_{r_1^i,G} + F\left(X_{r_2^i,G} - X_{r_3^i,G}\right) + F\left(X_{r_4^i,G} - X_{r_5^i,G}\right) \tag{9.24e}$$

The indices $r_1^i, r_2^i, r_3^i, r_4^i, r_5^i$ are mutually exclusive integers generated randomly within the range $[1, NP]$, which are also different from the index i. These indices are arbitrarily created once for each mutant vector. For scaling the difference vector, a positive control parameter F (scaling factor) is used. $X_{\text{best},G}$ is the best individual vector with the best fitness value in the population at generation G. A good initial choice of F was 0.5. The effective range of F values have been suggested between 0.4 and 1.0.

Note that the smaller the differences between parameters of parent, the smaller the difference vectors and therefore, the perturbation. That indicates whenever the population gets close to the optimum; the step length decreases automatically. This is similar to the automatic step size control found in the standard evolution strategies.

9.3.4 Crossover

After the mutation phase, crossover or recombination operation is applied to each pair of the target vector $X_{i,G}$ and its corresponding mutant vector $V_{i,G}$ to generate a trial vector:

$U_{i,G} = \left(u_{i,G}^1, u_{i,G}^2, \ldots, u_{i,G}^D\right)$. In order to increase the diversity of the perturbed parameter vectors, crossover is introduced. In the basic version, DE employ the binomial (uniform) crossover defined as follows:

$$u_{i,G}^j = \begin{cases} u_{i,G}^j \text{ if } \left(\text{rand}_j[0,1] \le CR\right) \text{ or } \left(j = j_{\text{rand}}\right) \\ x_{i,G}^j \qquad\qquad \text{otherwise} \end{cases}, \quad j = 1, 2, \ldots, D \tag{9.25}$$

In Eq. (9.25), CR, the crossover rate is a user-defined constant in the range [0,1], which controls the fraction of parameter values copied from the mutant vector. The parameter j_{rand} is a randomly selected integer within the range $[1,D]$. The binomial crossover operator copies the jth parameter of the mutant vector $V_{i,G}$ to the corresponding element in the trial vector $U_{i,G}$ if $\text{rand}_j\,[0,1] \le \text{CR}$ or $j = j_{\text{rand}}$. Otherwise, it is copied from the corresponding target vector $X_{i,G}$. There is an another exponential crossover operator, in which the parameters of trial vector $U_{i,G}$ are inherited from the corresponding mutant vector $V_{i,G}$ starting from a randomly selected parameter index till the first time $\text{rand}_j\,[0,1] > \text{CR}$. The remaining parameters of the trial vector $U_{i,G}$ are copied from the corresponding target vector $X_{i,G}$. The first reasonable attempt of selecting CR value can be 0.1. However, since the large CR value can accelerate convergence, the value of 0.9 for CR may also be a good initial choice if the problem is near unimodal or fast convergence is preferred. The condition $j = j_{\text{rand}}$ is introduced to make sure that the trial vector $U_{i,G}$ will be different from its corresponding target vector $X_{i,G}$ by at least one parameter. DE's exponential crossover operator is functionally equivalent to the circular two-point crossover operator.

9.3.5 Selection

The trial vector $U_{i,G+1}$ is compared to the target vector $X_{i,G}$ using the greedy criterion to decide whether the trial vector should become a member of generation $G + 1$ or not. Whenever the vector $U_{i,G+1}$ yields a smaller cost function value than $X_{i,G}$, then $X_{i,G+1}$ is set to $U_{i,G+1}$; or else, the old value $X_{i,G}$ is retained. The selection operation can be expressed as follows:

$$X_{i,G+1} = \begin{cases} U_{i,G} \text{ if } f\left(U_{i,G}\right) \leq f\left(X_{i,G}\right) \\ X_{i,G} \qquad \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, N \tag{9.26}$$

Generation after generation the above 3 steps are repeated until some specific termination criteria are met.

9.4 Simulated Annealing

Since its inception in the early 80's, [Kirkpatrick *et al.* (1983)] simulated annealing (SA) has been applied to various combinatorial and numerical optimization problems. SA is a generic probabilistic metaheuristic algorithm for the global optimization problem. This method has been developed based on the thermal annealing of critically heated metals. A solid metal becomes molten at a high temperature and the atoms of the melted metal move freely with respect to each other. However, as the temperature is decreased, these movements of atoms get restricted. When the temperature decreases, the atoms tend to get ordered and finally form crystals with the minimum possible internal energy. This process of crystals formation essentially depends on the cooling rate; fast cooling may result defects inside the material. The material may not be able to get the crystalline shape if the temperature of the molten metal is diminished at a very fast rate; instead, it may achieve a polycrystalline shape with a higher energy state compared to that of the crystalline shape. Therefore, the temperature of the molten metal should be decreased at a sluggish and controlled rate to make sure proper solidification with a highly ordered crystalline shape that corresponds to the lowest energy state (internal energy). This process of slow rate cooling is known as annealing [Rao, (2009)]. SA is a random-search method, which exploits an analogy between the way in which a molten metal cools and freezes into a lowest energy crystalline state and the search for a minimum in a more general system. Using the cost function in place of the energy and defining configurations by a set of parameters $\{x_i\}$, it is simple by means of the Metropolis procedure to generate a population of configurations of a given optimization problem at some efficient temperature. This temperature is just a control parameter in the same units as the cost function. [Kirkpatrick *et al.* (1983)]. Iterative improvement, usually employed to these problems, is much like the microscopic rearrangement processes represented by statistical mechanics, where the cost function was playing the role of energy. Though, accepting only rearrangements that lower the cost function of the system is like extremely rapid quenching from high temperatures to $T = 0$, so it should not be surprising that resulting solutions are usually metastable. The Metropolis procedure from statistical mechanics gives a generalization of iterative upgrading in which controlled uphill steps can likewise be incorporated in the search for a better solution [Kirkpatrick *et al.* (1983)].

Local minimization is no guarantee of global minimization. Therefore, a fundamental concern in global minimization is to avoid being stuck in a local minimum. There are two categories of methods known to overcome this difficulty in stochastic minimization: The first class constitutes the so-called two-phases methods; the second class is based on SA [Dekkers and Aarts (1991)]. The main aspect of SA algorithm is its ability to avoid being trapped in a local minimum. This is done letting the algorithm to accept not only better solutions however, also worse solutions (by allowing hill-climbing moves) with a given probability. [Chibante *et al.* (2010)] From the mathematical point of view, SA can be considered as a randomization tool that permits wrong-way movements during the search for the optimum through an adaptive acceptance/rejection criterion. This mechanism is of significance for treating effectively non-convex problems [Wei-zhong and Xi-Gang (2009)]. Relatively it is slow compared to deterministic optimization techniques using gradient based searches, branch and bound techniques and/or decomposition. Its primary weaknesses is the computational time complexity. However, its robustness and capability of handling discontinuous functions and discrete variables, where deterministic techniques fail, make it an essential tool for single and multiple objective optimization problems [Suman *et al.* (2010)].

There are two critical parameters in the cooling process that decide how amorphous or firm will be the result for the metal in its frozen condition. The first parameter is the initial temperature from which the cooling starts and the second one is the rate at which the temperature is decreasing. For any stochastic local search algorithms, some control parameters like initial temperature (T_0), cooling rate, are subjective to some extent and must be defined from an empirical basis. This shows that the algorithm must be adjusted in order to maximize its performance. [Shojaee *et al.* (2010)].

9.4.1 Procedure

Simulated annealing method follows the steps similar to the cooling of molten metal. Annealing procedure involves first 'melting' the system at a high temperature, and then decreasing the temperature gradually taking enough steps at each temperature to keep the system close to equilibrium, until the system reaches the ground state [Nourani and Andresen (1998)]. A typical procedure of SA is given below:

9.4.1.1 Algorithm of SA

This simulation procedure starts with initialization of vector X,(X_0) and temperature T, (T_0). Various researches [citation] show that parameters like number of iteration, cooling rate, and termination criteria (freezing temperature) have great influence on the SA algorithm performance. At each temperature level, system should reach equilibrium with minimum energy. For this purpose, a large number of iteration is required at each temperature level. This section elucidates how to estimate these parameters. Figure 9.9 shows the flowchart of the SA.

Step 1 Estimate the initial temperature (T_0), define number of iteration in each temperature step (n), number of cycles (k), freezing or final temperature (T_F), constant for annealing schedule (c)

Step 2 Guess an initial value (X_0), Calculate the value of the objective function $f(X_0)$

Step 3 Generate a new design point (X_{i+1}) in the vicinity of X_i, calculate the value of $f(X_{i+1})$; Set iteration counter $i = 1$, cycles counter $k = 1$.

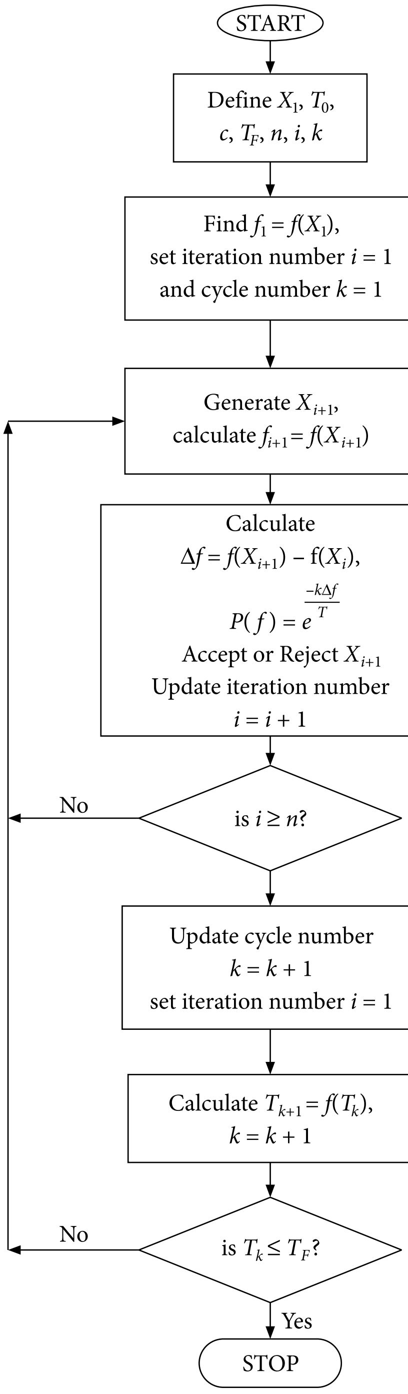
START
Define X_1, T_0, c, T_F, n, i, k
Find $f_1 = f(X_1)$, set iteration number $i = 1$ and cycle number $k = 1$
Generate X_{i+1}, calculate $f_{i+1} = f(X_{i+1})$
Calculate $\Delta f = f(X_{i+1}) - f(X_i)$, $P(f) = e^{\frac{-k\Delta f}{T}}$ Accept or Reject X_{i+1} Update iteration number $i = i + 1$
is $i \geq n$?
No
Update cycle number $k = k + 1$ set iteration number $i = 1$
Calculate $T_{k+1} = f(T_k)$, $k = k + 1$
is $T_k \leq T_F$?
No
Yes
STOP

Fig. 9.9 Flowchart of simulated annealing

Step 4 Find the value of D$f = f(X_{i+1}) - f(X_i)$, use Metropolis criterion (Eq. (9.27)) to decide whether to accept or reject the new design point.

$$P(f) = \begin{cases} e^{\frac{-k\Delta f}{T}} & \Delta f > 0 \\ 1 & \Delta f \le 0 \end{cases} \tag{9.27}$$

if probability $P(f)$ is within the acceptable range, accept the new design point; otherwise reject it and go back to the previous design point.

Step 5 Go to Step 3, with $i = i + 1$; Repeat the following sub-steps for n times (n is called the epoch length).

Step 6 If the given stop condition ($T_k \le T_F$) is satisfied, stop. Otherwise, reduce the temperature $T_{k+1} = f(T_k)$ according to annealing schedule and $k = k + 1$; and go to Step 3.

Theoretically, only if the inner loop computing (Step 3 – Step 5) is saturated (n is sufficiently large) to let the system reach to an equilibrium state, the global optimal can be expected. However, in practical use, only a limited number of configurations can be created at each temperature level, and then there exist a trade-off between computational cost and the quality of the solution [Wei-zhong and Xi-Gang (2009)].

9.4.1.2 Estimation of initialization temperature (T_0)

The initial temperature (T_0) plays a crucial role during simulated annealing process. Because T_0 controls the acceptance rule defined by Eq. (9.27) as well as the convergence of the algorithm. If T_0 is too large, it requires a large number of temperature reduction steps to converge. Conversely, if the initial temperature is selected to be very small, the search procedure may be incomplete in the sense that it might fail to thoroughly explore the design space to locate the global minimum before convergence. If the initial temperature is not high enough, atoms of the molten metal would not have full freedom to rearrange their positions in a very regular minimum energy structure. In a typical execution of SA, the initial temperature is set sufficiently high to enable the algorithm to move off a local minimum. Research shows that there is not any deterministic criterion to set the initial pseudo-temperature in the literature; formulation of the initial temperature is characteristics of the particular problem [Shojaee *et al.* (2010)].

According to the fundamental concepts of SA, non-improver solutions are accepted in the primary iterations with high probability. Pao *et al.* considered an initial temperature so that the initial acceptance rate is about 70 per cent [Pao *et al.* (1999)]. Thompson and Bilbro set the probability of accepting to 0.75 for initial temperature setting of continuous problems. Then, the following probability distribution is solved to find T_0 [Thompson and Bilbro, (2005)]:

$$p = \exp\left(\Delta E / T\right) \tag{9.28}$$

where DE is the average cost of the random solutions plus its standard deviation. Hao Chen *et al.* set the initial temperature so that the initial acceptance probability for an average uphill move is 0.97 [Chen *et al.* (1998)]. Feng-Tse Lin, *et al.* proposed an Annealing-Genetic approach and use the following formula [Lin *et al.* (1993)]:

$$T_0 = \frac{\Delta E}{\left(\text{Population size}/2\right)} \tag{9.29}$$

where DE is the difference between the highest cost and the lowest cost found for the first generation of the randomly generated population.

Parameter setting for SA is problem dependent, and it is best accomplished using trial and error. Furthermore, researches show that SA algorithms are highly sensitive to parameters, and their performances are largely dependent on fine tuning of the parameters [Wong and Constantinides, (1998)].

For calculating the initial temperature of the example 9.5, we have followed a simple method. The average value of the cost function in the first generation was considered as initial temperature.

9.4.1.3 Perturbation mechanism

The perturbation mechanism is the process to create new solutions from the current solution. A randomly generated perturbation of the current system configuration is applied so that a trial configuration is obtained. In other words, it is a process to explore the neighborhood of the current solution creating small variations in the current solution. At the beginning of the process, the temperature is high and strong perturbations are applied to easily jump over high peaks in search of the global minima. Although applying too much perturbation is useless and should be avoided.

SA is normally used in combinatorial problems where the parameters being optimized are integer numbers. In an application where the parameters vary continuously, the exploration of neighborhood solutions can be made as discussed here. A solution s is defined as a vector $s = (x_1,x_2,\ldots,x_n)$ representing a point within the search space. A new solution is generated using a vector $\sigma = \left(\sigma_1, \sigma_2, \ldots, \sigma_n\right)$ of standard deviations to create a perturbation from the current solution. A neighbor solution is then generated from the current solution by:

$$x_{i+1} = x_i + N\left(0, \sigma_i\right) \tag{9.30}$$

where $N\left(0, \sigma_i\right)$ is a random Gaussian number with zero mean and σ_i standard deviation.

To improve the performance of simulated annealing method, it is usual to spend some time at each temperature by conducting a fixed number of iterations before reducing the temperature. Consider K be the number of iterations executed at each temperature, then Equation (1) can be written as

$$X_i = \left\{x_{1,i}, x_{2,i}, \ldots, x_{M,i}\right\},\ i = 1, 2, \ldots \tag{9.31}$$

where i represents the number of perturbations applied to the solution, and $x_{1,r}$ is the value of x_1 after it is has been perturbed i-times. Hence, at the end of temperature T_1, the number of perturbations applied to the solution is K, and X_K represents the solution at the end of this temperature.

For finding the next design point (X_{i+1}) in the example 9.5, we have considered the range [X_i – 5, X_i + 5] and random numbers.

9.4.1.4 Cooling scheduling

The major drawback of SA is that, its massive requirement of computation time. This can be overcome using improved cooling schedule or annealing schedule. The sequence of temperatures and the number of rearrangements of the $\{x_i\}$ attempted to reach equilibrium at each temperature can be considered an annealing schedule. [Kirkpatrick *et al.* (1983)]. By controlling the temperature T, we control the probability of accepting a hill climbing move (a move that results in a positive Ax) and, therefore, the exploration of the state space. An annealing or cooling schedule controls how quickly the temperature T reduces from high to low values, as a function of time or iteration counts [Shojaee *et al.* (2010)]. The efficient annealing schedule, which lowers the temperature at every step and keeps the system in quasi-equilibrium at all times, is derived from a new quasi-equilibrium criterion.

The most common cooling schedule is the geometric rule for temperature variation:

$$T_{i+1} = sT_i \tag{9.32}$$

with $s < 1$, good results have been reported in literature when s is in the range [0.8, 0.99] [Fouskakis and Draper, (2002)]

The problem dependent nature of parameter setting for SA and its sensitivity to parameters restrict the robustness and effectiveness of SA algorithms. SA possesses a formal proof of convergence to the global optima. This convergence proof relies on a very slow cooling schedule with an initial condition of a sufficiently large temperature and let it decay by following relation:

$$T_k = \frac{T_0}{\log(k)} \tag{9.33}$$

where k is bound by the number of iterations [Ingber and Rosen, (1992)]. While this cooling schedule is impractical, it identifies a useful trade-off where longer cooling schedules tend to lead to better quality solutions.

A logarithmic cooling scheme introduced by Geman and Geman [Geman and Geman (1984)] is as follows:

$$T(t) = \frac{c}{\log(t+d)} \tag{9.34}$$

where d is usually set equal to one. As the only existence theorem, it has been established that for c being greater than or equal to the largest energy barrier in the problem, this schedule will lead the system to the global minimum state in the limit of infinite time. Hajek (1988) [Hajek (1988)] proved that exponential cooling could not guarantee convergence.

9.4.1.5 Termination criteria

The process is to have converged when the current value of temperature T is very small or when changes in the function value (Δf) are observed to be sufficiently small. The stochastic search process is terminated whenever (a) a given total number of steps have been carried out, (b) the expected number of acceptances for a given number of trials has not been achieved, (c) the annealing temperature drops below the freezing point. In most the cases, condition (c) is used as termination criteria. The calculation is terminated and the best solution is taken as the optimal solution when annealing temperature becomes equal or smaller than T_{end}.

The system is considered frozen and the process is terminated when no significant improvement is expected by further lowering the temperature. At this point, the current state of the system is the solution to the optimization problem.

Example 9.5

Find the minimum of the function using simulated annealing.

$$f(X) = 1200(2x_1x_3 + 2x_2x_3) + 2500x_1x_2 + 500\left(\frac{1000}{x_1x_2x_3}\right) + 100\left(\frac{1000}{10x_1x_2x_3}\right)$$

The condition for solving this problem is $x_1, x_2, x_3 \geq 0$

Solution

We considered only two iterations for each temperature level. However, for practical application more iteration is required to reach thermal equilibrium.

The cooling schedule used for this problem is $T_{i+1} = 0.5T_i$

$$X^{(1)} = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^T,\ X^{(2)} = \begin{bmatrix}1 & 5 & 10\end{bmatrix}^T,\ X^{(3)} = \begin{bmatrix}10 & 5 & 5\end{bmatrix}^T,\ X^{(4)} = \begin{bmatrix}7 & 6 & 5\end{bmatrix}^T$$

$$f^{(1)} = 517.3,\ f^{(2)} = 166.7,\ f^{(3)} = 307.04,\ f^{(4)} = 263.43$$

the average value of $f^{(1)}, f^{(2)}, f^{(3)}$ and $f^{(4)}$ is 313.6175

so, we are considering 313.62 as initial temperature

Step 1 Initial design point $X_1 = [3\ 4\ 5]^T$ and $f_1 = 122.50$

Step 2 Random numbers generated using simulink MATLAB are as follows

$$r_1 = 0.38,\ r_2 = 0.57,\ r_3 = 0.51$$

Generate a new design point in the vicinity (± 5.00) of the current design point.
Choose the ranges of x_1, x_2, and x_3; (–2,8), (–1,9), and (0,10)

now calculate $u_1 = -2 + 0.38[8 - (-2)] = 1.8$

$$u_2 = -1 + 0.57[9 - (-1)] = 4.7$$

$$u_3 = 0 + 0.51[10 - 0] = 5.1$$

which gives $X_2 = \begin{bmatrix}1.8\\4.7\\5.1\end{bmatrix}$ and $f_2 = 112.53$

value of for $p = 2$ $\quad \Delta f = f_2 - f_1 = 112.53 - 12250 = -9.97$

The new temperature, $T_2 = 0.5 \times 313.62 = 156.81$

$$r_1 = 0.68,\ r_2 = 0.45,\ r_3 = 0.46$$

the new ranges of x_1, x_2, and x_3 (–3.2,6.8), (–0.3,9.7), and (0.1,10.1)

$$u_1 = -3.2 + 0.68\left[6.8 - (-3.2)\right] = 3.6$$

$$u_2 = -0.3 + 0.45\left[9.7 - (-0.3)\right] = 4.2$$

$$u_3 = 0.1 + 0.46\left[10.1 - 0.1\right] = 4.5$$

which gives $\quad X_3 = \begin{bmatrix} 3.6 \\ 4.2 \\ 4.5 \end{bmatrix}$ and $f_3 = 129.534$

value of $\quad \Delta f = f_3 - f_2 = 129.534 - 112.53 = 17.006$

with $\quad k = 1,\ P\left[X_3\right] = e^{-\Delta f / T} = e^{-17.006/156.81} = 0.897$

so, we will accept the design point with a probability of 0.897

now we have to find X_4 in the neighbor of X_3 with random numbers

$$r_1 = 0.43,\ r_2 = 0.47,\ r_3 = 0.46$$

Calculated values for other iterations is given in the table below

Table 9.4 Calculated values of different iteration

P	i	T_i ($c = 0.5$)	r	u	X_i	$E_i = f_i = f(X_i)$	$\Delta E = E_{i+1} - E_i$	$P[E_{i+1}]$	accept/ reject X_i
1	0	313.62			[3,4,5]	122.50			
	1		[0.38,0.57,0.51]	[1.8,4.7,5.1]	[1.8,4.7,5.1]	112.53	−9.97	1	accept
2	1	156.81	[0.68,0.45,0.46]	[3.6,4.2,4.5]	[3.6,4.2,4.5]	129.534	17.006	0.897	accept
	2		[0.43,0.47,0.46]	[2.9,3.9,4.1]	[2.9,3.9,4.1]	106.185	−23.349	1	accept
3	1	78.405	[0.48,0.42,0.44]	[2.7,3.1,3.5]	[2.7,3.1,3.5]	87.054	−19.131	1	accept
	2		[0.31,0.83,0.71]	[0.8,6.4,5.6]	[0.8,6.4,5.6]	127.355	40.301	0.598	accept
4	1	39.202	[0.63,0.22,0.31]	[2.1,3.6,3.7]	[2.1,3.6,3.7]	87.749	−39.606	1	accept
	2		[0.93,0.65,0.35]	[6.4,5.1,2.2]	[6.4,5.1,2.2]	149.422	61.673	0.207	reject
5	1	19.601	[0.61,0.56,0.36]	[3.2,4.2,2.3]	[3.2,4.2,2.3]	90.946	3.197	0.8495	accept
	2		[0.45,0.13,0.53]	[2.7,2.5,2.6]	[2.7,2.5,2.6]	78.383	−12.564	1	accept

The graph (Fig. 9.10) shows that how the probability of accepting the new design point changes with temperature. At low temperature, probability $P[E]$ is low. We reject the design point when probability is below certain value. In Example 9.5 (Table: 9.4) shows that design point [6.4,5.1,2.2] has been rejected as the $P[E]$ is 0.207.

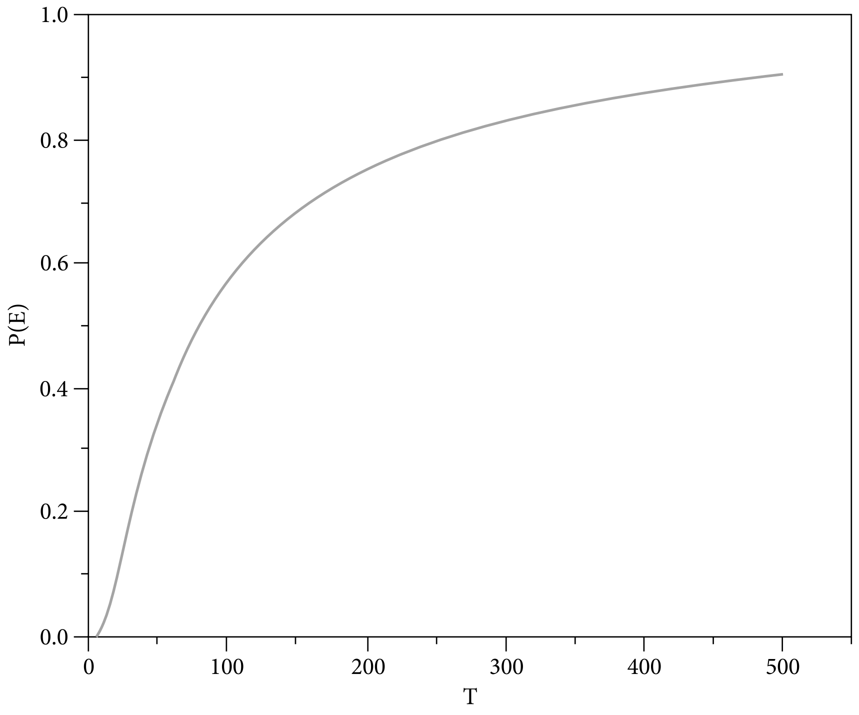

Fig. 9.10 Probability of accepting vs temperature plot

9.4.2 Applications of SA in chemical engineering

As simulated annealing is able to find the global optimum points, it is very useful in process engineering. Sankararao and Gupta [Sankararao and Gupta (2007)] used SA for optimizing fluidized bed catalytic cracking unit. They used multiobjective optimization for this problem. Some researchers [Kaczmarski and Antos (2006)] have used SA for chromatographic separation.

Summary

- This chapter discusses different non-traditional optimization techniques like genetic algorithm, particle swarm optimization, differential evolution, and simulated annealing. Genetic algorithm and differential evolution imitate the steps of genetic evolution, the particle swarm optimization method was developed based on the social behavior of a bird flocks, school of fish etc. Simulated annealing is a metaheuristic algorithm, which mimics the process of thermal annealing of critically heated metals. These methods are applicable for both continuous and discrete optimization. They are also able to find the global optimizer. However, we should be careful about the termination criteria of these algorithms.

Exercise

9.1 Covert these decimal number to binary

a) 25 b) 3 c) 98

9.2 How do you calculate the fitness of a chromosomes during genetic reproduction?

9.3 Write the algorithm for solving the problem

$$\min f(X) = 80\left(x_1^2 + x_2\right)^2 + \left(1 - x_1\right)^2$$

subject to $X \in [0,5]$

9.4 Compare the crossover and mutation operation.

9.5 solve the given problem using PSO method

$$f(X) = 1200\left(2x_1x_3 + 2x_2x_3\right) + 2500x_1x_2 + 500\left(\frac{1000}{x_1x_2x_3}\right) + 100\left(\frac{1000}{10x_1x_2x_3}\right)$$

9.6 Compare different neighborhood topologies for PSO method.

9.7 Write down the algorithm for differential evolution for solving the problem on 3.

9.8 Solve the optimization problem using SA

$$\min f(X) = 100\left(x_1^2 + x_2\right)^2 + \left(1 - x_1\right)^2$$

subject to $X \in [0,7]$

9.9 Show 3 iterations for solving the problem below

$$\min f(X) = -12x_1 - 7x_2 + x_2^2$$

subject to $-2x_1^4 - x_2 + 2 = 0$

$0 \le x_1 \le 3$, $0 \le x_2 \le 3$

9.10 Discuss the perturbation mechanism of simulated annealing.

9.11 Compare different cooling schedule of simulated annealing.

9.12 Discuss the termination criteria of simulated annealing. How the result depends on the final temperature?

9.13 What are the pros and cons of GA and SA? Can we conceive of a framework that combines the best of both worlds?

9.14 Are Genetic Algorithms useful if we do not have a full understanding of the objective function?

9.15 If two successive generations are identical then the Genetic Algorithm has found the optimal solution. Discuss if this statement is correct.

References

Babu, B. V. and Angira, R. 2006. *Modified Differential Evolution (MDE) for Optimization of Non-linear Chemical Processes,* Computers Chemical Engineering, 30(6–7): 989–1002.

Bhaskar, V., Gupta, S. K. and Ray, A. K. 2000. *Applications of Multi-objective Optimization in Chemical Engineering,* Rev Chemical Engineering, 16: 1–54.

Bratton, D. and Kennedy, J. 2007. *Defining a Standard for Particle Swarm Optimization,* IEEE Swarm Intelligence Symposium, pp. 120–27.

Chen, H., Flann, N. S. and Watson, D. W. 1998. *Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm,* IEEE Transactions on Parallel and Distributed Systems, 9(2):126–136.

Chibante, R. Araújo A. and Carvalho A. 2010. *Parameter Identification of Power Semiconductor Device Models using Metaheuristics,* Simulated Annealing Theory with Applications (edited by Rui Chibante), Chapter 1.

Clerc, M. 1999. *The Swarm and the Queen: Towards a Determininistic and Adaptive Particle Swarm Optimization,* 'Congress on Evolutionary Computation' (CEC99), pp. 1951–57.

Clerc, M. and Kennedy, J. 2002. *The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space,* 'IEEE Transactions on Evolutionary Computation', 6: 58–73.

Curteanu, S. Leon, F. 2008. *Optimization Strategy Based on Genetic Algorithms and Neural Networks Applied to a Polymerization Process,* 'International Journal of Quantum Chemistry', vol. 108, 617–30.

Davis, L. 1991. *Handbook of Genetic Algorithms,* New York: Van Nostrand Reinhold, pp. 11–96.

Dekkers, A. and Aarts, E. *Global Optimization and Simulated Annealing;* Mathematical Programming 50(1991): 367–93.

Edgar, T. F., Himmelblau, D. M. and Lasdon, L. S. 2002. *Optimization of Chemical Processes,* McGraw-Hill Inc.

Fouskakis, D. and Draper, D. 2002. *Stochastic Optimization: A Review, International Statistical Review,* 70(3): 315–49.

Garrard, A. and Fraga, E. S. 1998. *Mass Exchange Network Synthesis using Genetic Algorithms,* Computers Chemical Engineering, 22, pp. 1837.

Geman, S. and Geman, D. 1984. *Stochastic Relaxation, Gibbs Distributions, and Bayesian Restoration of Images IEEE Trans.* Pattern Anal. Mach. Intell., PAMI-66: 721–41.

Ghalia, M. B. *Particle Swarm Optimization with an Improved Exploration-Exploitation Balance,* IEEE, vol. 978-1-4244-2167-1/08/ 2008.

Goldberg, D. E. 1989. *Genetic Algorithms in Search Optimization and Machine Learning,* Addison-Wesley Pub. Co., pp. 147–260.

Hajek, B. 1988. *Cooling Schedules for Optimal Annealing Math Oper Res* 13: 311–29 op cit Azencott.

Handbook of Evolutionary Computation (1997). IOP Publishing Ltd. and Oxford University Press, release 97/1.

Heppner, F. and Grenander, U. A. 1990. *Stochastic Nonlinear Model for Coordinated Bird Flocks.* In S. Krasner, Ed., The Ubiquity of Chaos, Washington. DC: AAAS Publications.

Herrera, F., Lozano, M., Verdegay, J. L. 1998. *Tackling Real Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis.* Artificial Intelligence Review, 12: 265.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, MI.

Ingber, L. and Rosen, B. 1992. *Genetic Algorithms and Very Fast Simulated Reannealing:* 'A Comparison, Mathematical Computer Modeling', vol. 16, No. 11, pp. 87–100.

Jung, J. H., Lee, C. H. and Lee, I-B. 1998. *A Genetic Algorithm for Scheduling of Multiproduct Batch Processes,* Comp. Chem. Eng., 22: 1725.

Kaczmarski, K. and Antos, D. 2006. *Use of Simulated Annealing for Optimization of Chromatographic Separations;* Acta Chromatographica, No. 17: 20–45.

Kennedy, J. and Eberhart, R. 1995. *Particle Swarm Optimization.* 'In Proceedings of IEEE International Conference on Neural Networks', vol. IV, pp. 1942–48, Perth, Australia.

Kirkpatrick, S., Gelatt, C. D. Vecchi, M. P. Jr. 1983. *Optimization by Simulated Annealing; Science, New Series,* vol. 220, No. 4598. pp. 671–80.

Lin, F-T., Kao, C-Y. and Hsu C-C. 1993. *Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems,* IEEE Transactions on Systems, Man, and Cybernetics, 23(6).

Löeh, T., Schulz, C. and Engell, S. 1998. *Sequencing of Batch Operations for Highly Coupled Production Process: Genetic Algorithms vs. Mathematical Programming,* Comp. Chem. Eng., 22: S579.

Mayer, D. G. 2002. *Evolutionary algorithms and agricultural systems,* Boston: Kluwer Academic Publishers, pp. 107.

Nandasana, A. D. Ray, A. K. and Gupta, S. K. 2003. *Application of the Non-dominated Sorting Genetic Algorithm (NSGA) in Chemical Engineering,* Int J Chem Reactor Eng 1: 1.

Nourani, Y. and Andresen, B. *A Comparison of Simulated Annealing Cooling Strategies;* J. Phys. A: Math. Gen., 31(1998): 8373–85.

Pao, D. C. W. Lam, S. P. and Fong, A. S. 1999. *Parallel Implementation of Simulated Annealing Using Transaction Processing,* IEE Proc-Comput. Digit. Tech., 146(2): 107–13.

Price, K. V. Differential Evolution: *A Fast and Simple Numerical Optimizer,* 0–7803–3225–3–6/96 1996 IEEE.

Qin, A. K. Huang, V. L. and Suganthan, P. N. *Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization,* IEEE Transactions on Evolutionary Computation, 13(2): 2009.

Rao, S. S. 2009. *Engineering Optimization: Theory and Practice* (4e), John Wiley and Sons.

Reynolds, C. W., *Flocks, Herds, and Schools: A Distributed Behavioral Model.* Computer Graphics, vol. 21(1987): 25–34.

Sankararao, B. Gupta, S. K. *Multi-objective Optimization of an Industrial Fluidized-bed Catalytic Cracking Unit (FCCU) using Two Jumping Gene Adaptations of Simulated Annealing;* Computers and Chemical Engineering, 31(2007): 1496–1515.

Shojaee, K., Shakouri, H. G. and Taghadosi, M. B. 2010. *Importance of the Initial Conditions and the Time Schedule in the Simulated Annealing a Mushy State SA for TSP,* Simulated Annealing Theory with Applications (edited by Rui Chibante); Chapter 12.

Shopova and Vaklieva-Bancheva (2006) Shopova, E. G.; Vaklieva-Bancheva N. G. Comp Chem Eng 2006, 30: 1293.

Storn, R. Price, K. 1997. *Differential Evolution – a Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces.* Journal of Global Optimization 11: 341–359.

Storn R. *On the Usage of Differential Evolution for Function Optimization.* Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS). (1996): 519–523.

Suman, B. Hoda, N. and Jha, S. *Orthogonal Simulated Annealing for Multiobjective Optimization;* Computers and Chemical Engineering 34(2010): 1618–31.

Thompson, D. R. and Bilbro, G. L. 2005. *Sample-Sort Simulated Annealing,* IEEE Transactions on Systems, Man, and Cybernetics—PART B: Cybernetics, 35(3): 625–632.

Trelea, I. C. 2003. *The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection.* Information Processing Letters, 85: 317–25.

Valle, Y. del, Venayagamoorthy G. K., Mohagheghi S., Hernandez J–C., and Harley R. G. 2008. *Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems,* IEEE Transactions on Evolutionary Computation, 12(2).

Weber, T. and Bürgi, H. B. 2002. *Determination and Refinement of Disordered Crystal Structures Using Evolutionary Algorithms in Combination with Monte Carlo Methods;* Acta Cryst. A58, pp. 526–40.

Wei-zhong A., Xi-Gang Y., *A Simulated Annealing-based Approach to the Optimal Synthesis of Heat-Integrated Distillation Sequences,* Computers and Chemical Engineering, 33(2009): 199–212.

Wong, K. L., Constantinides A. G. 1998. *Speculative Parallel Simulated Annealing with Acceptance Prediction,* Electronics Letters, 34(3): 312–13.